Dynamic Load Balancing in Parallelization of Equation-based Models

Mahder Gebremedhin

Programing Environments Laboratory (PELAB), IDA
Linköping University
mahder.gebremedhin@liu.se

Annual OpenModelica Workshop
2016-02-01
Linköping, Sweden
Overview

- Introduction
- Extracting Parallelism
- Task System Library
- Performance
- Future work
FourBitBinaryAdder: Dependency Task Graph

Original

- 1122 tasks
- 1360 edges
Automatic Parallelization

Improving the compiler

- Design and implementation of new automatic parallelization support for the OpenModelica compiler.
- Design and implementation of customizable task system handling library.
- Multiple clustering and scheduling options.
- Targeting shared-memory multi-core architectures.
Dependency Analysis

\[ f_1(x_1, x_2, t) = 0 \]
\[ f_2(x_3, t) = 0 \]
\[ f_3(x_1, x_3, x_4, t) = 0 \]
\[ f_4(x_3, x_5, t) = 0 \]
\[ f_5(x_1, x_4, x_5, t) = 0 \]
\[ f_6(x_6, t) = 0 \]
\[ f_7(x_6, x_7, t) = 0 \]
x_3 := g_2(t)

x_5 := g_4(x_3, t)

g_3(x_1, x_3, x_4, t) = 0

g_5(x_1, x_4, x_5, t) = 0

x_2 := g_1(x_1, t)

x_6 := g_6(t)

x_7 := g_7(x_6, t)
**Strongly Connected Components**

\[ x_3 := g_2(t) \]
\[ x_5 := g_4(x_3, t) \]
\[ \{x_1, x_4\} := g_{35}(x_3, x_5, t) \]
\[ x_2 := g_1(x_1, t) \]
\[ x_6 := g_6(t) \]
\[ x_7 := g_7(x_6, t) \]
Decoupled Systems

Systems \( \{g_6, g_7\} \) and \( \{g_2, g_4, g_{35}, g_1\} \) are not connected and can potentially run in parallel.

Transmission Line Modeling (TLM)

- Introduces *delays* to the system.
- Better decoupling by eliminating some dependencies in each time step.

Coarse Grained Parallelization

- Find all decoupled systems.
- Balance these systems.
- Evaluate them simultaneously.
TLM and Decoupled Systems

Problems with the approach
- Most models are heavily connected, i.e. limited decoupling.
- Improving decoupling with TLM requires modification to existing models.

Problems with the implementation
- Implemented as part of the normal code-generation runtime system.
- Complicates development process.

New approach
- Task graph based representation of whole system.
- Library based implementation.
From Equation Systems to Task Graphs

Directed Acyclic Graphs

\[ G = (\vec{V}, \vec{E}, c) \]
The Task Systems Library

What?
- Generic C++ template task system library.
  - Tasks
  - Clusters
  - Clustering algorithms
  - Scheduling algorithms
  - Profiling and execution

Dependencies
- Boost
- Intel Threading Building Blocks (TBB)
Tasks and Clusters

Tasks
- Abstract task representation that can be customized.
- Define dependency and execution rules.

Clusters
- Every vertex is a cluster.
- Originally each cluster contains one task.
- Tasks in a single cluster are executed sequentially and in order.
Clustering Algorithms

Cost Oblivious
- Merge Single Parent (MSP)
- Merge Level Parents (MLP).

Cost Based
- Merge Children Recursive (MCR)
- Merge Level for Cost (MLC)
Profiling and Cost Estimation

Static Cost Estimation

- User provided cost values.
- Suitable for handling tasks that are executed only once.
- For simulation environments
  - Can be estimated by traversing abstract syntax trees or internal representation.

Limitations

- Not accurate.
- Some tasks are not easy to estimate, e.g. function calls, loops...
- Costs vary on different architectures.
Profiling and Cost Estimation

Dynamic Cost Estimation

- Execute once and record.
- Suitable for simulation environments.
  - Simulations execute systems repeatedly.

Current implementation

- First time step of simulation used for profiling.
- Clustering, Scheduling and subsequent evaluations use this profiling information.
- Should be done periodically.
Schedulers

Collection of clustering algorithms.
Profiling.
Executors and synchronizations.

Available Schedulers
Level Scheduler.
TBB Flow Graph Based Scheduler.
Level Scheduler

Clustering
- Merge Children Recursive.
- Merge Level for Cost.

Executor
- StepSync
  - Execute all tasks in the same level.
  - Synchronize.

Level Scheduler Class

```cpp
template<typename TaskType>
struct LevelScheduler :
  StepSync < TaskType >
, MCR
, MLC
> {};
```
Wrapper for TBB flow graph

- Profile the system.
- Perform Clustering.
- Construct flow graph and execute.

Why not directly create flow graph

- Clustering improves performance by reducing overhead.
- Consistency in external interface.
FourBitBinaryAdder: Dependency Task Graph Before Clustering

Original
- 1122 tasks
- 1360 edges
FourBitBinaryAdder: Dependency Graph after Clustering for Level Scheduler

8-way

After Merge Children Recursive
- 569 tasks
- 620 edges

After Merge Level for Cost: 8
- 27 tasks
- 121 edges

4-way

After Merge Level for Cost: 4
- 18 tasks
- 72 edges
Performance Measurements

Measurement Setup

- 64-bit Intel(R) Xeon(R) W3565 CPU with 4 cores at 3.2 GHz.
- Simulation 0 to 1 second.
- Default OpenModelica Solver (DASSL)
- Only the ODE system is parallelized for each model.

Estimated Level Scheduler Speedup

Ratio of the sequential cost to the ideal parallel cost.
Branching Dynamic Pipes (Fluid)

![Bar chart showing speedup for different schedulers and thread counts.](image)

- **Estimated Level Scheduler**
- **Achieved Level Scheduler**
- **Achieved Flow Graph Scheduler**

### Speedup
- **2 threads**
  - Estimated Level Scheduler: 1.88
  - Achieved Level Scheduler: 1.73
  - Achieved Flow Graph Scheduler: 2.37

- **4 threads**
  - Estimated Level Scheduler: 3.82
  - Achieved Level Scheduler: 2.13
  - Achieved Flow Graph Scheduler: 2.44
Future work

- More clustering and scheduling algorithms.
- Better adaptive rescheduling with continuous dynamic scheduling.
- Extensive testing and comparison.
Thank You!